On the Discriminating Power of Passivation and Higher-Order Interaction

Marco Bernardo, Davide Sangiorgi, Valeria Vignudelli

On the Discriminating Power of Passivation and Higher-Order Interaction

Marco Bernardo
University of Urbino

Davide Sangiorgi
University of Bologna and INRIA

Valeria Vignudelli
University of Bologna and INRIA

Abstract

This paper studies the discriminating power offered by *higher-order concurrent* languages, and contrasts this power with those offered by *higher-order sequential* languages (à la λ-calculus) and by *first-order concurrent* languages (à la CCS). The concurrent higher-order languages that we focus on are Higher-Order π-calculus (HOπ), which supports higher-order communication, and an extension of HOπ with *passivation*, a simple higher-order construct that allows one to obtain location-dependent process behaviours.

The comparison is carried out by providing embeddings of first-order processes into the various languages, and then examining the resulting contextual equivalences induced on such processes. As first-order processes we consider both ordinary Labeled Transition Systems (LTSs) and Reactive Probabilistic Labeled Transition Systems (RPLTSs).

The hierarchy of discriminating powers so obtained for RPLTSs is finer than that for LTSs. For instance, in the LTS case, the additional discriminating power offered by passivation in concurrency is captured, in sequential languages, by the difference between the call-by-name and call-by-value evaluation strategies of an extended typed λ-calculus.

Categories and Subject Descriptors F.3.1 [*Logics and Meaning of Programs*]: Specifying and Verifying and Reasoning about Programs—logics of programs; F.3.2 [*Logics and Meaning of Programs*]: Semantics of Programming Language—operational semantics

General Terms Theory

Keywords higher-order pi-calculus, lambda-calculus, CCS, contextual equivalence, passivation, probabilistic processes

1. Introduction

Higher-order languages have been widely studied in functional programming, following the λ-calculus. In a higher-order language, variables may be instantiated with terms of the language. When multiple occurrences of the variable exist, this mechanism results in the possibility of copying the terms of the language.

Higher-order concurrency combines functional programming and concurrent programming: the ability of exchanging values, common in concurrency, is enhanced by allowing values to include terms of the language itself, the distinguishing feature of functional languages. Calculi of this kind include CHOCS [30] and the Higher-Order π-calculus [25].

An important extension to concurrent higher-order languages concerns distribution. This is usually achieved by means of constructs for expressing and operating on locations. As a consequence, the observable behaviour of a system of processes depends not only on the behaviour of the constituent processes, but also on the locations in which these processes are run. This can have a deep impact on the behavioural theory and algebraic laws for the language.

One of the simplest constructs that show these phenomena is *passivation*. Passivation offers the capability of capturing the content of a certain location, and then restarting the execution in a different context. The semantics of passivation has been the subject of a number of papers, usually in extensions of the Higher-Order π-calculus [15, 19–21, 24]. Passivation is also featured in the Homer calculus [10] and the M-calculus [26]; a similar construct appears in the Seal calculus [4] and in Acute [29]. Passivation has also been advocated to support run-time system updates, fault recovery and fault tolerance (by providing the basis for mechanisms for checkpointing computations and replicating them), and to support adaptive behaviours.

The goal of this paper is to shed light into the discriminating power offered by *higher-order concurrent* languages, and contrasting this power with those offered by *higher-order sequential* languages (which are deprived of all concurrency) and by *first-order concurrent* languages (which are deprived of all higher-order features). The comparison is carried out by considering embeddings of first-order processes into the languages, and then examining the equivalences induced by the resulting contextual equivalences on the first-order processes. In other words, the discriminating power of a language refers to the existence of appropriate contexts of the language that are capable of separating the behaviours of first-order processes.

The higher-order sequential languages are typed λ-calculi extended with a kell construct [27] akin to a store location of imperative λ-calculi. The λ-calculi offer constructs for reading the kell, overriding it, and, if the kell contains a process, for consuming such a process (i.e., performing observations on the process actions). The higher-order concurrent languages are HOπ, which allows higher-order communication, and HO$\pi_{\mathtt{pass}}$, an extension of HOπ with passivation (similar to the languages in [15, 21, 24]). Both languages also admit first-order communications, to be able to interact with the embedded first-order processes. The first-order concurrent language that we consider is CCS$^-$, a CCS-like calculus.

The λ-calculi also allow us to observe the inability for a process to perform a certain action. In concurrency, this possibility is referred to as *action refusal*. For a thorough comparison, we therefore also consider both restrictions of the λ-calculi without the refusal observation (though at the price of allowing computations that may get stuck) and extensions of HOπ, HO$\pi_{\mathtt{pass}}$, and CCS$^-$ with the refusal capability.

Concerning the tested first-order processes, embedded into the above languages, we consider both ordinary LTSs (which exhibit internal nondeterminism, e.g., a process that has multiple transitions with the same label) and reactive probabilistic LTSs (which exhibit probabilistic choices on transitions with the same label, hence admit no internal nondeterminism). We simply call LTSs the former and RPLTSs the latter.

We show that, on LTSs, the difference between the discriminating power of HOπ and HO$\pi_{\mathtt{pass}}$ is captured, in the λ-calculi, through the difference between the call-by-name and call-by-value evaluation strategies, both with and without refusals.

The correspondence between the HO$\pi_{\mathtt{pass}}$ calculi and the call-by-value λ-calculi appears robust, and is maintained in all scenarios examined. The same does not hold between the HOπ calculi and the call-by-name λ-calculi, whose correspondence breaks on RPLTSs. The case of RPLTSs is more involved also when we consider the first-order language CCS$^-$. For instance, the discriminating power of CCS$^-$ is strictly in between that of the call-by-name λ-calculus and HOπ. In contrast, the three languages are equally discriminating on LTSs.

We also discuss variations of the above settings. In languages with locations, communication may or may not be affected by spatial proximity. This is the difference between global vs. local communications. This difference is important for the semantics of the languages but, as we shall see, does not impinge on their discriminating power.

The contextual equivalences that we consider are may-like (a test, i.e., a context, is successful on a process if there is at least one successful computation). We also discuss the contextual preorders, and 'must' forms of success (all computations are successful). We isolate a few scenarios in which, surprisingly, the may and must forms of contextual equivalence coincide.

There are analogies between our results on the contextual equivalences induced by higher-order languages on ordinary LTSs and results in the literature on the equivalences on LTSs that characterize the coarsest congruences contained in trace equivalence for operators whose operational rules comply with certain *rule formats*. Some of these formats allow negative premises in the rules, with which refusals may be encoded, or allow rules in which an argument of an operator may end up, in the derivative of the rule, within a predefined context; when the context is polyadic, this yields a form of copying. In higher-order languages, in contrast, copying is achieved through the variable binding mechanisms of the languages. Passivation or, in the λ-calculi, call-by-value, are necessary to obtain the discriminating power of powerful formats such as GSOS [2] and tyft/tyxt [11] (which give ready simulation equivalence and simulation equivalence, respectively).

Rule formats for probabilistic processes include [1, 5, 17], where the emphasis is on ensuring congruence properties for bisimilarity. Testing of reactive probabilistic processes is studied in [16], obtaining an equivalence strictly coarser than bisimilarity, though the comparison with the equivalences induced by our contextual equivalences is unclear. Testing equivalences in which the testers are allowed to make copies of the tested RPLTSs have been studied by Larsen and Skou [18] and Van Breugel et al. [31] to recover bisimilarity. Our characterizations of bisimilarity in higher-order languages exploit these results: in one direction of the proofs, we essentially implement in higher-order languages the tests needed in [18, 31] to distinguish non-bisimilar RPLTSs. In contrast, copies of the tested process are not possible in our CCS-like languages.

Paper structure: Section 2 presents background material: LTSs, RPLTSs, equivalences on them. Section 3 considers the embeddings of LTSs and RPLTSs into λ-calculi. Section 4 shows the syntax and operational semantics of the concurrent languages (CCS- and HOπ-like), whose discriminating power is studied in Sections 5 and 6. Section 7 discusses variations of the scenarios examined. Section 8 reports conclusions and possible future work.

Notation: In examples, we sometimes use a CCS-like notation, with prefixing and choice, to describe the processes of an LTS or RPLTS.

2. First-Order Processes and Equivalences

In this section, we introduce the two LTS-like models used in the paper to formalize respectively fully nondeterministic processes (LTSs) and reactive probabilistic processes (RPLTSs), and we recall a number of behavioural equivalences for these first-order processes. Then, we define contextual equivalences induced on such processes by their embedding into algebraic languages.

2.1 Fully Nondeterministic Processes

The behaviour of a fully nondeterministic process can be represented through a labeled transition system.

DEFINITION 2.1. A *labeled transition system* (LTS) is a triple $(S, A, \longrightarrow)$ where S is a countable set of states (usually called processes), A is a countable set of transition-labeling actions, and $\longrightarrow \subseteq S \times A \times S$ is a transition relation. The LTS is *image-finite* if $\{P' \in S \mid P \xrightarrow{a} P'\}$ is finite for all $P \in S$ and $a \in A$. ■

We will characterize the contextual equivalences induced on LTSs in terms of simulation equivalence [22], ready simulation equivalence [2, 18], trace equivalence [3], failure equivalence [3], and failure-trace equivalence [23, 32].

DEFINITION 2.2. Let $(S, A, \longrightarrow)$ be an LTS and $\mathcal{R}$ be a relation over S. Relation $\mathcal{R}$ is a *simulation* if, whenever $(P_1, P_2) \in \mathcal{R}$, then for all $a \in A$ it holds that for each $P_1 \xrightarrow{a} P_1'$ there exists $P_2 \xrightarrow{a} P_2'$ such that $(P_1', P_2') \in \mathcal{R}$. Relation $\mathcal{R}$ is a *ready simulation* if, additionally, $P_1 \xrightarrow{a}$ implies $P_2 \xrightarrow{a}$. Processes $P_1, P_2 \in S$ are *simulation equivalent* ($P_1 \sim_{\mathrm{S}} P_2$) – resp., *ready simulation equivalent* ($P_1 \sim_{\mathrm{RS}} P_2$) – if there exist two simulations – resp., ready simulations – $\mathcal{R}$ and $\mathcal{R}'$ such that $(P_1, P_2) \in \mathcal{R}$ and $(P_2, P_1) \in \mathcal{R}'$. ■

A *trace* is an element σ of A^*. We let $\mathcal{CC}(P, \sigma)$ denote the set of σ-compatible computations from P, i.e., the computations starting from P and labeled with σ. We call *failure pair* an element $\varphi \in A^* \times 2^A$ composed by a trace σ and a refusal set F. We denote by $\mathcal{FCC}(P, \varphi)$ the set of φ-compatible computations from P, i.e., the computations starting from P and labeled with σ whose last state has no outgoing transitions labeled with an action in F. We call *failure trace* an element $\phi \in (A \times 2^A)^*$ given by a sequence of $n \in \mathbb{N}$ pairs of the form (a_i, F_i). We indicate with $\mathcal{FTCC}(P, \phi)$ the set of ϕ-compatible computations from P, i.e., the computations starting from P and labeled with trace $a_1 \ldots a_n$ such that the state reached after the i-th step, $1 \leq i \leq n$, has no outgoing transitions labeled with an action in the refusal set F_i.

DEFINITION 2.3. Let $(S, A, \longrightarrow)$ be an LTS. Processes $P_1, P_2 \in S$ are:

- *trace equivalent*, written $P_1 \sim_{\mathrm{Tr}} P_2$, if $\mathcal{CC}(P_1, \sigma) \neq \emptyset \iff \mathcal{CC}(P_2, \sigma) \neq \emptyset$ for all $\sigma \in A^*$;

- *failure equivalent*, written $P_1 \sim_{\rm F} P_2$, if $\mathcal{FCC}(P_1,\varphi) \neq \emptyset \iff \mathcal{FCC}(P_2,\varphi) \neq \emptyset$ for all $\varphi \in A^* \times 2^A$;
- *failure-trace equivalent*, written $P_1 \sim_{\rm FTr} P_2$, if $\mathcal{FTCC}(P_1,\phi) \neq \emptyset \iff \mathcal{FTCC}(P_2,\phi) \neq \emptyset$ for all $\phi \in (A \times 2^A)^*$. ■

For instance, consider the processes defined in Examples 3.1 and 3.2: P and P' are failure-trace (and hence trace) equivalent but not simulation equivalent, while P' is simulation but not failure-trace equivalent to P''; processes Q, Q' are not ready simulation equivalent, but they are both simulation and failure-trace equivalent.

2.2 Reactive Probabilistic Processes

A reactive probabilistic process features probabilistic choices but no internal nondeterminism. Its behaviour can be described as a variant of LTS [18]. The set of discrete probability distributions over a set S is denoted by $Distr(S)$.

DEFINITION 2.4. A *reactive probabilistic labeled transition system* (RPLTS) is a triple $(S, A, \longrightarrow)$ where S is a countable set of states (usually called processes), A is a countable set of transition-labeling actions, and $\longrightarrow \subseteq S \times A \times Distr(S)$ is a transition relation such that $P \xrightarrow{a} \Delta_1$ and $P \xrightarrow{a} \Delta_2$ imply $\Delta_1 = \Delta_2$ for all $P \in S$ and $a \in A$. ■

Given a transition $P \xrightarrow{a} \Delta$, a process $P' \in S$ is not reachable from P via that a-transition if $\Delta(P') = 0$, otherwise it is reachable with probability $p = \Delta(P')$. The reachable states form the support of Δ, i.e., $supp(\Delta) = \{P' \in S \mid \Delta(P') > 0\}$. The choice of the action to be performed by an RPLTS process is made by the external environment, and then the target state is selected internally, purely probabilistically.

We will compare the equivalences induced on RPLTSs with probabilistic bisimulation equivalence [18] and probabilistic variants of trace equivalence and failure-trace equivalence.

DEFINITION 2.5. Let $(S, A, \longrightarrow)$ be an RPLTS. An equivalence relation $\mathcal{B}$ over S is a *probabilistic bisimulation* if, whenever $(P_1, P_2) \in \mathcal{B}$, then for all $a \in A$ it holds that $P_1 \xrightarrow{a} \Delta_1$ implies $P_2 \xrightarrow{a} \Delta_2$ with $\Delta_1(S') = \Delta_2(S')$ for all $S' \in S/\mathcal{B}$. Processes $P_1, P_2 \in S$ are *probabilistic bisimilar*, written $P_1 \sim_{\rm PB} P_2$, if there exists a probabilistic bisimulation $\mathcal{B}$ such that $(P_1, P_2) \in \mathcal{B}$. ■

In the RPLTS setting, each state-to-state step $P \xrightarrow{a}\!\!\!\!\rightarrow P'$ of a computation is derived from a state-to-distribution transition $P \xrightarrow{a} \Delta$.

DEFINITION 2.6. Let $\mathcal{L} = (S, A, \longrightarrow)$ be an RPLTS and $P, P' \in S$. The sequence

$$c \equiv P_0 \xrightarrow{a_1}\!\!\!\!\rightarrow P_1 \xrightarrow{a_2}\!\!\!\!\rightarrow P_2 \ldots P_{n-1} \xrightarrow{a_n}\!\!\!\!\rightarrow P_n$$

is a *computation* of $\mathcal{L}$ of length n from $P = P_0$ to $P' = P_n$ if for all $i = 1, \ldots, n$ there exists a transition $P_{i-1} \xrightarrow{a_i} \Delta_i$ such that $P_i \in supp(\Delta_i)$, with $\Delta_i(P_i)$ being the execution probability of step $P_{i-1} \xrightarrow{a_i}\!\!\!\!\rightarrow P_i$ conditioned on the selection of transition $P_{i-1} \xrightarrow{a_i} \Delta_i$ of $\mathcal{L}$ at state P_{i-1}. We denote by $\mathcal{C}_{\rm fin}(P)$ the set of finite-length computations from P. ■

Given a computation $c \in \mathcal{C}_{\rm fin}(P)$, its conditional execution probability $prob(c)$ can be defined as the product of the conditional execution probabilities of the individual steps of c. This notion is lifted to a set $\mathcal{C} \subseteq \mathcal{C}_{\rm fin}(P)$ of identically labeled computations by letting $prob(\mathcal{C}) = \sum_{c \in \mathcal{C}} prob(c)$.

DEFINITION 2.7. Let $(S, A, \longrightarrow)$ be an RPLTS. Processes P_1, $P_2 \in S$ are:

- *probabilistic trace equivalent*, written $P_1 \sim_{\rm PTr} P_2$, if $prob(\mathcal{CC}(P_1,\sigma)) = prob(\mathcal{CC}(P_2,\sigma))$ for all $\sigma \in A^*$;
- *probabilistic failure-trace equivalent*, written $P_1 \sim_{\rm PFTr} P_2$, if $prob(\mathcal{FTCC}(P_1,\phi)) = prob(\mathcal{FTCC}(P_2,\phi))$ for all $\phi \in (A \times 2^A)^*$. ■

In Example 3.4, P is not probabilistic trace equivalent to P'' and is probabilistic trace (but not probabilistic failure-trace) equivalent to P'. Finally, Q and Q' are probabilistic failure-trace equivalent, but they are not probabilistic bisimilar.

2.3 Contextual May-Equivalence

Given a set of processes as states of an LTS or RPLTS $\mathcal{L}$, and an algebraic language AL (i.e., generated by a grammar), the equivalence induced by AL equates the $\mathcal{L}$ processes P_1 and P_2 if $C[P_1]$ and $C[P_2]$ behave the same for all contexts C of AL. Here 'behave the same' is formalized as in ('may') contextual equivalence: $C[P_1]$ is as successful as $C[P_2]$ with respect to a special success observation, indicated with ω. The context C is an AL-expression with a *single* occurrence of the hole $[\cdot]$ in it.

In the remainder of the paper, $\mathcal{L}$ is always the LTS or RPLTS of tested first-order processes, and P, Q range over $\mathcal{L}$ processes. Moreover, in these tested processes each transition represents a visible action, i.e., there is a corresponding coaction with which the action can synchronize and produce a reduction; and the actions available for $\mathcal{L}$ do not include the success signal ω. We write $\mathtt{AL}(\mathcal{L})$ for the extension of AL with the $\mathcal{L}$ processes; i.e., the grammar for AL is extended so to include the $\mathcal{L}$ processes. In a language $\mathtt{AL}(\mathcal{L})$, reductions are represented as τ-transitions $\xrightarrow{\tau}$ (or simply $\longrightarrow$, in λ-calculi). Each language AL used will have constructs for testing the action capabilities of $\mathcal{L}$ processes; thus, the set of action names for $\mathcal{L}$ is supposed to appear in the grammar for AL. We emphasize that probabilities may appear in the tested $\mathcal{L}$ processes, but they may *not* appear in the AL languages that test the processes.

The operational semantics of $\mathtt{AL}(\mathcal{L})$ will be based on different LTS-like models depending on the nature of $\mathcal{L}$. A finite-length computation c from a term $M \in \mathtt{AL}(\mathcal{L})$ is *successful* if each step of c is labeled with τ, the last state of c can perform ω, and no preceding state of c can perform ω. We denote by $\mathcal{SC}(M)$ the set of successful computations from M. In the fully nondeterministic case, when $\mathcal{L}$ is an LTS, the semantic model underlying $\mathtt{AL}(\mathcal{L})$ is again an LTS.

DEFINITION 2.8. Let $\mathcal{L}$ be an LTS, P_1 and P_2 two processes of $\mathcal{L}$, and AL an algebraic language. In $\mathtt{AL}(\mathcal{L})$:

- P_1 is *contextually may-less* than P_2, written $P_1 \leq^{\mathcal{L}}_{\mathtt{AL}} P_2$, if $\mathcal{SC}(C[P_1]) \neq \emptyset \implies \mathcal{SC}(C[P_2]) \neq \emptyset$ for all contexts C of AL.
- P_1 is *contextually may-equivalent* to P_2, written $P_1 \simeq^{\mathcal{L}}_{\mathtt{AL}} P_2$, if $P_1 \leq^{\mathcal{L}}_{\mathtt{AL}} P_2$ and $P_2 \leq^{\mathcal{L}}_{\mathtt{AL}} P_1$. ■

In the case that $\mathcal{L}$ is an RPLTS, the definition of contextual equivalence is more involved because the semantic model underlying $\mathtt{AL}(\mathcal{L})$ is a nondeterministic and probabilistic LTS (NPLTS for short), an extension of RPLTS in which internal nondeterminism is allowed (i.e., the constraint at the end of Definition 2.4 is removed).

For NPLTS computations, using probabilities conditioned on actions is not sufficient, because a state may have several outgoing transitions labeled with the same action. As is standard in the NPLTS setting, we thus resort to *resolutions*. A resolution of a state M of an NPLTS $\mathcal{W}$ is the result of a possible way of resolving nondeterminism, which is obtained by unfolding from M the graph structure underlying $\mathcal{W}$ while selecting from the current state at most one of its outgoing transitions.

DEFINITION 2.9. Let $\mathcal{W} = (S, A, \longrightarrow)$ be an NPLTS and $M \in S$. An NPLTS $\mathcal{Z} = (Z, A, \longrightarrow_{\mathcal{Z}})$ is a *resolution* of M if there exists a state correspondence function $corr_{\mathcal{Z}} : Z \to S$ such that $M = corr_{\mathcal{Z}}(z_M)$, for some $z_M \in Z$, and for all $z \in Z$ it holds that:

- If $z \xrightarrow{a}_{\mathcal{Z}} \Delta$, then $corr_{\mathcal{Z}}(z) \xrightarrow{a} \Delta'$ with $corr_{\mathcal{Z}}$ being injective over $supp(\Delta)$ and $\Delta(z') = \Delta'(corr_{\mathcal{Z}}(z'))$ for all $z' \in Z$.
- If $z \xrightarrow{a_1}_{\mathcal{Z}} \Delta_1$ and $z \xrightarrow{a_2}_{\mathcal{Z}} \Delta_2$, then $a_1 = a_2$ and $\Delta_1 = \Delta_2$.

$\mathcal{Z}$ is *maximal* if, for all $z \in Z$, whenever z has no outgoing transitions, then $corr_{\mathcal{Z}}(z)$ has no outgoing transitions either. We respectively denote by $Res(M)$ and $Res_{\max}(M)$ the sets of resolutions and maximal resolutions of M. ■

As $\mathcal{Z} \in Res(M)$ is fully probabilistic, the probability $prob(c)$ of executing $c \in \mathcal{C}_{\text{fin}}(z_M)$ is the product of the (no longer conditional) execution probabilities of the individual steps of c. This notion is lifted to $\mathcal{C} \subseteq \mathcal{C}_{\text{fin}}(z_M)$ by letting $prob(\mathcal{C}) = \sum_{c \in \mathcal{C}} prob(c)$ whenever none of the computations in $\mathcal{C}$ is a proper prefix of one of the others.

The contextual equivalence defined below is inspired by [8, 13, 28, 33]. Intuitively, P_1 is worse than P_2 if, for all contexts C, the maximum probability of reaching success in an arbitrary maximal resolution of $C[P_1]$ is not greater than the maximum probability of reaching success in an arbitrary maximal resolution of $C[P_2]$. To correctly quantify success, we restrict ourselves to $Res_{\tau,\max}(C[P])$, the set of maximal resolutions obtained from $C[P]$ by forbidding the execution of actions not resulting in interactions (i.e., τ actions).

DEFINITION 2.10. Let $\mathcal{L}$ be an RPLTS, P_1 and P_2 be two processes of $\mathcal{L}$, and $\mathtt{AL}$ be an algebraic language. Indicating with ρ_i the predicate $\mathcal{Z}_i \in Res_{\tau,\max}(C[P_i])$ for $i = 1, 2$, we say that in $\mathtt{AL}(\mathcal{L})$:

- P_1 is *contextually may-less* than P_2, written $P_1 \leq^{\mathcal{L}}_{\mathtt{AL}} P_2$, if for all contexts C of $\mathtt{AL}$ it holds that:
$$\bigsqcup_{\rho_1} prob(\mathcal{SC}(z_{C[P_1]})) \;\leq\; \bigsqcup_{\rho_2} prob(\mathcal{SC}(z_{C[P_2]}))$$
- P_1 is *contextually may-equivalent* to P_2, written $P_1 \simeq^{\mathcal{L}}_{\mathtt{AL}} P_2$, if $P_1 \leq^{\mathcal{L}}_{\mathtt{AL}} P_2$ and $P_2 \leq^{\mathcal{L}}_{\mathtt{AL}} P_1$. ■

We sometimes abbreviate 'contextual may equivalence' as 'contextual equivalence' or even 'may equivalence'.

3. λ-Calculi

3.1 Syntax

Figure 1 shows the syntax of the (typed) λ-calculus $\mathtt{K}\Lambda$ into which we embed the processes of a (first-order) LTS or RPLTS $\mathcal{L}$. The grammar of the language resulting from the embedding, $\mathtt{K}\Lambda(\mathcal{L})$, has therefore the additional production

$$M := \ldots \;\Big|\; P$$

where P is an $\mathcal{L}$ process; moreover, in the action test, r is supposed to range over the actions in $\mathcal{L}$. A λ-term M is evaluated in an environment that may contain another λ-term N, written $\langle N\,;\,M\rangle$. The environment will correspond, in the concurrent calculi of the next section, to a location, and therefore, following the concurrency terminology, we call such an environment the *kell*. The language includes a construct `pass` for obtaining the content of the kell, and a construct $\langle N\,;\,M\rangle$ for creating the kell. The action-test construct allows us to check whether the process that the kell evaluates to can perform a certain action. The remaining constructs are common constructs of typed λ-calculi. We assume that the set of constants includes the boolean values `true` and `false` and the unit value $\star$. For lack of space we omit the type system (simply-typed with recursive types; for simplicity, the expressions in the kell may only have process type). In contrast to the usual operators for kells in concurrency, in $\mathtt{K}\Lambda$ the kell is persistent: `pass` reads the kell without destroying it. This simplifies the typing system and does not affect the results presented.

Reduction is defined on terms that are *closed* (i.e., without free variables) and *equipped with the kell*, i.e., of the form $\langle M_1\,;\,M_2\rangle$. As a consequence, the contexts in which the $\mathcal{L}$ processes are tested have the form $\langle M\,;\,C\rangle$ or $\langle C\,;\,M\rangle$.

3.2 Fully Nondeterministic Processes

We consider both *call-by-name* and *call-by-value* reduction strategies. We call $\mathtt{K}\Lambda_{\mathrm{N}}(\mathcal{L})$ the call-by-name language, $\mathtt{K}\Lambda_{\mathrm{V}}(\mathcal{L})$ its call-by-value version, as usual omitting $\mathcal{L}$ when referring to the pure languages (without $\mathcal{L}$ processes). When $\mathcal{L}$ is an LTS, a reduction step has the form $\langle M_1\,;\,M_2\rangle \longrightarrow \langle M_1'\,;\,M_2'\rangle$, saying that the evaluation of M_2 with kell M_1 produces a new term M_2' with kell M_1'. The rules for reduction are in Figure 2 (in rule EVALKELL, Q is some predefined specific process from $\mathcal{L}$ that is used to initialize the kell; it does not matter which particular Q is used for this – anyone would do). In $\mathtt{K}\Lambda_{\mathrm{N}}(\mathcal{L})$, only the functional part of an application is evaluated, hence rule BETA-V and the production VC for evaluation contexts are omitted. In $\mathtt{K}\Lambda_{\mathrm{V}}(\mathcal{L})$, both the function and the argument of an application are evaluated, hence rule BETA-N is omitted. In all these languages, although the operators of the λ-calculi themselves are sequential, nondeterministic computations are possible because the process in the kell may present internal nondeterminism. As usual, $\Longrightarrow$ is the reflexive and transitive closure of $\longrightarrow$.

In the call-by-name calculus $\mathtt{K}\Lambda_{\mathrm{N}}(\mathcal{L})$, during a computation an $\mathcal{L}$ process may be moved around, may be copied, and may be placed into the kell. However, once placed into the kell for evaluation, the process cannot be stopped *and* later re-evaluated. In call-by-value, by contrast, the PASS rule may be used by the argument of a function, and then the process so obtained may be passed onto the function; as a consequence, the process may later be evaluated. This gives us more sophisticated process tests than call-by-name. Example 3.1 shows how the terms in $\mathtt{K}\Lambda_{\mathrm{N}}$ and $\mathtt{K}\Lambda_{\mathrm{V}}$ can test the existence of 'barbed traces' in the behaviour of a process placed in the kell, and how some tests are available only in call-by-value.

EXAMPLE 3.1. Below, a test is encoded in $\mathtt{K}\Lambda_{\mathrm{V}}$ as a thunked boolean expression $\lambda.M$ and $(\lambda.M)\star$ is its 'unthunking' (thunking is useful in composition of tests). A test $\lambda.M$ is successful on a process P if there is a run in which `true` is produced, i.e., $\langle P\,;\,(\lambda.M)\star\rangle \Longrightarrow \langle N\,;\,\mathtt{true}\rangle$ for some N.

$$\begin{array}{rcl}
\mathrm{T}_a & \stackrel{\text{def}}{=} & \lambda.\,\mathtt{if}\ a?\ \mathtt{then\ true\ else\ false} \\
\mathrm{T}_{\not a} & \stackrel{\text{def}}{=} & \lambda.\,\mathtt{if}\ a?\ \mathtt{then\ false\ else\ true} \\
\underline{\mathrm{Seq}} & \stackrel{\text{def}}{=} & \lambda x.\lambda y.\lambda.\,\mathtt{if}\ x\star\ \mathtt{then}\ y\star\ \mathtt{else\ false} \\
M_1 & \stackrel{\text{def}}{=} & \underline{\mathrm{Seq}}\ \mathrm{T}_a\ (\,\underline{\mathrm{Seq}}\ \mathrm{T}_{\not a}\ \mathrm{T}_b\,) \\
\underline{\mathrm{And}} & \stackrel{\text{def}}{=} & \lambda x.\lambda y.\lambda.((\lambda z.\ \ \mathtt{if}\ x\star \\
 & & \qquad\qquad \mathtt{then}\ \langle z\,;\,y\star\rangle\ \mathtt{else\ false})\mathtt{pass}) \\
M_2 & \stackrel{\text{def}}{=} & \underline{\mathrm{Seq}}\ \mathrm{T}_a\ (\,\underline{\mathrm{And}}\ \mathrm{T}_b\ \mathrm{T}_c\,) \\
M_3 & \stackrel{\text{def}}{=} & \underline{\mathrm{Seq}}\ \mathrm{T}_a\ (\,\underline{\mathrm{And}}\,(\,\underline{\mathrm{Seq}}\ \mathrm{T}_b\ \mathrm{T}_c\,)(\,\underline{\mathrm{Seq}}\ \mathrm{T}_b\ \mathrm{T}_{\not a}))
\end{array}$$

Test T_a checks the possibility of performing action a (i.e., $P \xrightarrow{a}$), whereas $\mathrm{T}_{\not a}$ checks its impossibility. Function $\underline{\mathrm{Seq}}$ composes the two argument tests sequentially. Thus M_1 checks the existence of an a-derivative that cannot perform c but can perform b (i.e., $P \xrightarrow{a} P'$ with $P' \not\xrightarrow{c}$ and $P' \xrightarrow{b}$, for some P'). Function $\underline{\mathrm{And}}$ makes the conjunction of the two argument tests. Thus M_2 checks the existence of an a-derivative that can perform both b and c, while

Terms:	M	$::=$	x	(variables)	$\mid$ $\texttt{if}\ M_1\ \texttt{then}\ M_2\ \texttt{else}\ M_3$	(if-then-else)
		$\mid$	$\lambda x.M$	(functions)	$\mid$ $\texttt{pass}$	(passivation)
		$\mid$	$M_1 M_2$	(applications)	$\mid$ $\langle M_1 \,;\, M_2\rangle$	(kell creation)
		$\mid$	c	(constants)	$\mid$ $r?$	(action test)

Figure 1. Syntax of $\mathtt{K}\Lambda$

$$\textsc{Beta-N}\ \frac{}{\langle N \,;\, (\lambda x.M_1)M_2\rangle \longrightarrow \langle N \,;\, M_1\{^{M_2}/_x\}\rangle} \qquad \textsc{Beta-V}\ \frac{V \text{ is a value}}{\langle N \,;\, (\lambda x.M)V\rangle \longrightarrow \langle N \,;\, M\{^{V}/_x\}\rangle}$$

$$\textsc{If1}\ \frac{}{\langle N \,;\, \texttt{if true then } M_1 \texttt{ else } M_2\rangle \longrightarrow \langle N \,;\, M_1\rangle} \qquad \textsc{If2}\ \frac{}{\langle N \,;\, \texttt{if false then } M_1 \texttt{ else } M_2\rangle \longrightarrow \langle N \,;\, M_2\rangle}$$

$$\textsc{NewKell}\ \frac{}{\langle N \,;\, \langle M_1 \,;\, M_2\rangle\rangle \longrightarrow \langle M_1 \,;\, M_2\rangle} \qquad \textsc{Pass}\ \frac{}{\langle M \,;\, \texttt{pass}\rangle \longrightarrow \langle M \,;\, M\rangle} \qquad \textsc{EvalKell}\ \frac{\langle Q \,;\, M\rangle \longrightarrow \langle Q' \,;\, M'\rangle}{\langle M \,;\, r?\rangle \longrightarrow \langle M' \,;\, r?\rangle}$$

$$\textsc{Act}\ \frac{P \xrightarrow{r} P' \text{ (in } \mathcal{L})}{\langle P \,;\, r?\rangle \longrightarrow \langle P' \,;\, \mathsf{true}\rangle} \qquad \textsc{RefAct}\ \frac{P \not\xrightarrow{r} \text{ (in } \mathcal{L})}{\langle P \,;\, r?\rangle \longrightarrow \langle P \,;\, \mathsf{false}\rangle} \qquad \textsc{EvCon}\ \frac{C \text{ is an evaluation context} \quad \langle N \,;\, M\rangle \longrightarrow \langle N' \,;\, M'\rangle}{\langle N \,;\, C[M]\rangle \longrightarrow \langle N' \,;\, C[M']\rangle}$$

Evaluation contexts: $C \ ::= \ [\cdot] \mid \texttt{if } C \texttt{ then } M_1 \texttt{ else } M_2 \mid CM \mid VC$

Values: $V \ :: \ = c \mid \lambda x.M \mid P \mid x$

Figure 2. The reduction rules of $\mathtt{K}\Lambda(\mathcal{L})$

M_3 checks the existence of an a-derivative with both a b-derivative that can perform c and a b-derivative that cannot perform c.

In $\mathtt{K}\Lambda_{\mathrm{N}}$, while $\underline{\mathrm{Seq}}$ and T_a, $\mathrm{T}_{\not a}$, M_1 have the same outcomes as in $\mathtt{K}\Lambda_{\mathrm{V}}$, function $\underline{\mathrm{And}}$ (and so also M_2, M_3) cannot be encoded. As a consequence, only the call-by-value calculi can separate

$$P \stackrel{\text{def}}{=} a.b + a.c \qquad \text{and} \qquad P' \stackrel{\text{def}}{=} a.(b + c) + P$$

When applied to P', test M_2 consumes an action a and then, in case the first a-branch of P' is taken, the whole expression reduces to

$$\langle b + c \,;\, (\lambda z.\, \texttt{if } \mathrm{T}_b \star \texttt{ then } \langle z \,;\, \mathrm{T}_c \star\rangle \texttt{ else false})\texttt{pass}\rangle$$

Now, term `pass` is not a value, hence in call-by-value it is evaluated and produces process $b + c$. Since processes are values, $b + c$ is substituted for the variable z. Thus $b + c$ is placed in a kell with which the test T_c is performed on the same process $b + c$, once the test T_b reports success. By contrast, in call-by-name `pass` is substituted for z before being evaluated, hence $b + c$ is lost before performing test T_c. ■

In λ-calculi, well-typed terms are supposed to produce computations that never get stuck. To maintain this property, we have to ensure that the action-test construct $a?$ returns a value even when the process in the kell is unable to perform the requested action a. That is, we are allowed to observe the inability for the kell process to perform a certain action, in concurrency referred to as *action refusal* and usually omitted. We therefore consider also variants of the above λ-calculi without the refusal capability. Formally, rule RefAct is omitted. Of course the price to pay is that computations from a well-typed term may get stuck. The call-by-name calculus without RefAct is called $\mathtt{K}\Lambda_{\mathrm{N-ref}}(\mathcal{L})$, whereas call-by-value without RefAct is $\mathtt{K}\Lambda_{\mathrm{V-ref}}(\mathcal{L})$.

Example 3.2. (We reuse P', M_1, M_3 from Example 3.1.) The processes P' and $P'' \stackrel{\text{def}}{=} a.(b + c)$ are distinguished in $\mathtt{K}\Lambda_{\mathrm{N}}$ and $\mathtt{K}\Lambda_{\mathrm{V}}$ (via the test M_1), but they are equivalent in $\mathtt{K}\Lambda_{\mathrm{N-ref}}$ and $\mathtt{K}\Lambda_{\mathrm{V-ref}}$. In contrast, only in $\mathtt{K}\Lambda_{\mathrm{V}}$ the processes

$$Q \stackrel{\text{def}}{=} a.b.c + a.b \qquad \text{and} \qquad Q' \stackrel{\text{def}}{=} a.(b.c + b)$$

can be separated via test M_3. ■

Theorem 3.3 summarizes the results on the discriminating power of the four λ-calculi when the embedded processes are fully nondeterministic. Definition 2.8 of contextual equivalence is adapted to the λ-calculi supposing that the reduction relation $\longrightarrow$ is labeled with τ and adding the rule:

$$\textsc{Omega}\ \frac{}{\langle N \,;\, \mathsf{true}\rangle \xrightarrow{\omega}}$$

Theorem 3.3. If $\mathcal{L}$ is an image-finite LTS, then:

1. $\simeq^{\mathcal{L}}_{\mathtt{K}\Lambda_{\mathrm{V}}} = \sim_{\mathrm{RS}}$ (ready simulation equivalence);
2. $\simeq^{\mathcal{L}}_{\mathtt{K}\Lambda_{\mathrm{N}}} = \sim_{\mathrm{FTr}}$ (failure trace equivalence);
3. $\simeq^{\mathcal{L}}_{\mathtt{K}\Lambda_{\mathrm{V-ref}}} = \sim_{\mathrm{S}}$ (simulation equivalence);
4. $\simeq^{\mathcal{L}}_{\mathtt{K}\Lambda_{\mathrm{N-ref}}} = \sim_{\mathrm{Tr}}$ (trace equivalence).

Proof [Sketch] The proofs are different for inductive and coinductive equivalences. For the inductive equivalences, we discuss failure-trace equivalence and $\mathtt{K}\Lambda_{\mathrm{N}}$. In one direction, one shows that for every failure trace ϕ there is a context C of $\mathtt{K}\Lambda_{\mathrm{N}}$ such that P has the failure trace ϕ iff $C[P]$ produces `true`.

For the opposite direction, suppose P and Q have the same failure traces, and suppose $C[P] \Longrightarrow\!\!\xrightarrow{\omega}$. We show that there is also a computation $C[Q] \Longrightarrow\!\!\xrightarrow{\omega}$, proceeding as follows. Consider the computation $C[P] \Longrightarrow\!\!\xrightarrow{\omega}$. During this computation the hole may get duplicated, and therefore the context may become polyadic. To analyze what happens to the context and the processes inside them during the computation, we adopt an annotated operational semantics (equivalent to the original one) in which, when a hole reaches a redex position (whereby the process in the hole may be evaluated), the hole is marked and the observations made on its process (the transition performed, the actions refused) are annotated. One then shows that the observations so annotated can only be failure traces. Hence process Q, which has the same failure traces as P, can mimic the computation $C[P] \Longrightarrow\!\!\xrightarrow{\omega}$.

As an example for the coinductive equivalences, we consider ready simulation equivalence and $\mathtt{K}\Lambda_{\mathrm{V}}(\mathcal{L})$. In one direction, suppose P is not ready simulated by Q. We exploit the assumption of image-finiteness for the LTS $\mathcal{L}$ and the inductive characterization of ready simulation via stratification approximants. Thus there is a minimal n such that P and Q are distinguished at the n-th approximant. Proceeding by induction on n we define a context of

the language that separates P and Q, in the sense that only $C[P]$ may reduce to $\mathtt{true}$.

For the opposite direction, one shows that the relation

$$\mathcal{R} \stackrel{\text{def}}{=} \{(C[P], C[Q]) \mid P \text{ is ready simulated by } Q\}$$

where C is a polyadic (and runnable) context, is a strong simulation on reductions (in the sense that if $M \ \mathcal{R} \ N$ and $M \longrightarrow M'$ then there is N' with $N \longrightarrow N'$ and $M' \ \mathcal{R} \ N'$). As a consequence, any successful computation from $C[P]$ may be mimicked by $C[Q]$. □

3.3 Reactive Probabilistic Processes

The reduction relation for $\mathtt{K}\Lambda(\mathcal{L})$ when $\mathcal{L}$ is an RPLTS is the expected probabilistic modification of the system for the nondeterministic case. A probability distribution reached from a process in the kell is propagated to a λ-term by using the rule

$$\textsc{Act} \quad \frac{P \xrightarrow{r} \Delta}{\langle P \,;\, r? \rangle \longrightarrow \langle \Delta \,;\, \overline{\mathsf{true}} \rangle}$$

For any term M, $\overline{M}$ is the Dirac probability distribution on M, i.e., $\overline{M}(M) = 1$ and $\overline{M}(M') = 0$ if $M \neq M'$. For all probability distributions Δ_1, Δ_2 on the terms of the language,

$$\langle \Delta_1 \,;\, \Delta_2 \rangle(M) \stackrel{\text{def}}{=} \begin{cases} \Delta_1(M_1) \cdot \Delta_2(M_2) & \text{if } M = \langle M_1 \,;\, M_2 \rangle \\ 0 & \text{otherwise} \end{cases}$$

Since the tested processes do not feature internal nondeterminism, all terms of $\mathtt{K}\Lambda_{\mathrm{N}}, \mathtt{K}\Lambda_{\mathrm{N-ref}}, \mathtt{K}\Lambda_{\mathrm{V}}$ and $\mathtt{K}\Lambda_{\mathrm{V-ref}}$ are reactive probabilistic processes. Hence, for any context C of these languages and for any reactive probabilistic process P the term $C[P]$ has only one maximal resolution on reductions. We write $prob(\mathcal{SC}(C[P]))$ to denote $prob(\mathcal{SC}(z_{C[P]}))$, where $z_{C[P]}$ is the state associated with $C[P]$ in the resolution.

EXAMPLE 3.4. Consider the processes and λ-terms defined below, where $\underline{\mathrm{Seq}}$, $\underline{\mathrm{And}}$, T_a, $\mathrm{T}_{\not a}$ are as in Example 3.1. In all four λ-calculi, the test M_1 separates between P and P'', since the resulting success probabilities are 0.5 and 0.25, respectively. The term M_2 distinguishes P and P' both in $\mathtt{K}\Lambda_{\mathrm{N}}$ and in $\mathtt{K}\Lambda_{\mathrm{V}}$, since $prob(\mathcal{SC}(\langle P \,;\, M_2 \star \rangle)) = 0.5$ and $prob(\mathcal{SC}(\langle P' \,;\, M_2 \star \rangle)) = 0$. The same processes are distinguished by M_3 in $\mathtt{K}\Lambda_{\mathrm{V-ref}}$, which shows that the refusal operator is not necessary if the tested processes can be copied. Finally, only in $\mathtt{K}\Lambda_{\mathrm{V-ref}}$ and $\mathtt{K}\Lambda_{\mathrm{V}}$ the test M_4 distinguishes Q and Q'.

$$\begin{aligned}
P &\stackrel{\text{def}}{=} a.((b + c.d) +_{0.5} (f + c.e)) \\
P' &\stackrel{\text{def}}{=} a.((b + c.e) +_{0.5} (f + c.d)) \\
P'' &\stackrel{\text{def}}{=} a.(((b + c.d) +_{0.5} c.d) +_{0.5} (f + c.e)) \\
Q &\stackrel{\text{def}}{=} a.(b.c +_{0.5} b) \quad Q' \stackrel{\text{def}}{=} a.b.(c +_{0.5} \mathbf{0}) \\
M_1 &\stackrel{\text{def}}{=} \underline{\mathrm{Seq}}\ \mathrm{T}_a\ (\mathrm{T}_b) \\
M_2 &\stackrel{\text{def}}{=} \underline{\mathrm{Seq}}\ \mathrm{T}_a\ (\underline{\mathrm{Seq}}\ \mathrm{T}_{\not f}\ (\underline{\mathrm{Seq}}\ \mathrm{T}_c\ \mathrm{T}_d)) \\
M_3 &\stackrel{\text{def}}{=} \underline{\mathrm{Seq}}\ \mathrm{T}_a\ (\underline{\mathrm{And}}\ \mathrm{T}_b\ (\underline{\mathrm{Seq}}\ \mathrm{T}_c\ \mathrm{T}_d)) \\
M_4 &\stackrel{\text{def}}{=} \underline{\mathrm{Seq}}\ \mathrm{T}_a\ (\underline{\mathrm{And}}\ (\underline{\mathrm{Seq}}\ \mathrm{T}_b\ \mathrm{T}_c)\ (\underline{\mathrm{Seq}}\ \mathrm{T}_b\ \mathrm{T}_c))
\end{aligned}$$

■

THEOREM 3.5. If $\mathcal{L}$ is an RPLTS, then:

1. $\simeq^{\mathcal{L}}_{\mathtt{K}\Lambda_{\mathrm{V}}} = \simeq^{\mathcal{L}}_{\mathtt{K}\Lambda_{\mathrm{V-ref}}} = \sim_{\mathrm{PB}}$ (probabilistic bisimilarity);
2. $\simeq^{\mathcal{L}}_{\mathtt{K}\Lambda_{\mathrm{N}}} = \sim_{\mathrm{PFTr}}$ (probabilistic failure trace equivalence);
3. $\simeq^{\mathcal{L}}_{\mathtt{K}\Lambda_{\mathrm{N-ref}}} = \sim_{\mathrm{PTr}}$ (probabilistic trace equivalence).

Proof [Sketch] For the inductive equivalences, the proof schemata are as in the nondeterministic case. For item (1), in one direction we exploit a result in [31]. Let $\mathbf{T}$ be the language of tests t defined as follows:

$$t ::= \omega \ \Big|\ r.t \ \Big|\ (t_1, t_2)$$

Let $\mathcal{P}(t, P)$ be the success probability for a test t on a reactive probabilistic process P. The test ω succeeds with probability 1, while $a.t$ checks whether P can perform an a-labeled transition, and then executes t. Formally, if $P \xrightarrow{a} \Delta$ then $\mathcal{P}(r.t, P) = \sum_{P' \in \mathrm{supp}(\Delta)} \Delta(P') \cdot \mathcal{P}(t, P')$, else $\mathcal{P}(r.t, P) = 0$. Finally, $\mathcal{P}((t_1, t_2), P) = \mathcal{P}(t_1, P) \cdot \mathcal{P}(t_2, P)$, i.e. (t_1, t_2) checks P's probability of passing both t_1 and t_2. On reactive probabilistic processes, $P \sim_{\mathrm{PB}} Q$ iff $\mathcal{P}(t, P) = \mathcal{P}(t, Q)$ for every test t in $\mathbf{T}$ [31]. We show that these tests are encodable in $\mathtt{K}\Lambda_{\mathrm{V-ref}}$ (some hints are provided by Example 3.1 and Example 3.4); i.e., for every test t in $\mathbf{T}$ there is a term M_t such that for every P, $\mathcal{P}(t, P) = prob(\mathcal{SC}(\langle P \,;\, M_t \rangle))$. Hence, if $P \not\sim_{\mathrm{PB}} Q$ then there is a context of $\mathtt{K}\Lambda_{\mathrm{V-ref}}$ that distinguishes P and Q. The same holds for the language $\mathtt{K}\Lambda_{\mathrm{V}}$, since it includes $\mathtt{K}\Lambda_{\mathrm{V-ref}}$.

For the other direction, the proof is in two steps, analogously to the case of nondeterministic processes: we first prove that if P, Q are probabilistic bisimilar $\mathcal{L}$ processes and C is a λ-context, then also $C[P]$ and $C[Q]$ are probabilistic bisimilar when the bisimulation game is only played on reductions and success (ω) transitions. We then prove that, if two λ-terms M, N are probabilistic bisimilar in this sense, then $prob(\mathcal{SC}(M)) = prob(\mathcal{SC}(N))$. A difficulty here is that M and N could have infinitely many successful computations, whose length is unbounded (e.g., Example 3.6). We then resort to a fixed-point approach. □

EXAMPLE 3.6. Let $P \stackrel{\text{def}}{=} a.(b +_{0.5} \mathbf{0})$ and M be

$$\lambda y.(\lambda x.\, \underline{\mathrm{Seq}}\ \mathrm{T}_a\ \lambda.\, \mathtt{if}\ b?\ \mathtt{then\ true\ else}\ \langle x \,;\, yy\star \rangle)\mathtt{pass}$$

In $\mathtt{K}\Lambda_{\mathrm{V}}$, the term $\langle P \,;\, MM\star \rangle$ with probability 0.5 reports success and with probability 0.5 becomes again $\langle P \,;\, MM\star \rangle$. Thus, there are infinitely many successful computations from $\langle P \,;\, MM\star \rangle$, and the overall success probability is 1. (Note that the typing of M requires recursive types.) ■

4. Concurrency: Syntax and Operational Rules

We present here the concurrent languages used to test the first-order processes taken from an LTS or RPLTS $\mathcal{L}$. This section gives the syntax and operational rules. The following two sections study the equivalences induced by the languages. To simplify the presentation, we assume that also first-order communications exchange values, namely the unit value $\star$. *Names* include channels $a, b, \ldots$ and *locations* $l, m, \ldots$. The operators are those common to CCS and higher-order π-calculi. The special prefix ω indicates *success* of a computation. We add the basic constructs of calculi with passivation, namely the kell $[\![M]\!]_l$ and the passivation prefix $\mathtt{pass}_l(x).M$. The *refusal* prefix $\widetilde{r}_l.M$, where l is a location containing $\mathcal{L}$ processes, succeeds if the process in l is unable to perform the action r. (The addition of other operators is discussed in Section 7.3.) Kells may be nested, and the kell structure is transparent with respect to communications. In the remainder, unless otherwise stated, all mentioned processes are supposed to be *closed* (without free variables). A channel or prefix is *first-order* or *higher-order* depending on whether the exchanged value is $\star$ or is a process. We sometimes abbreviate first-order prefixes $a(x).M$ and $\overline{a}\langle\star\rangle.M$ as $a.M$ and $\overline{a}.M$, and omit the trailing $\mathbf{0}$ in $\alpha.\mathbf{0}$.

The language with all operators, $\mathrm{HO}\pi_{\mathtt{pass,ref}}$, is given in Figure 3. The subset without the refusal prefix is $\mathrm{HO}\pi_{\mathtt{pass}}$; the subset without passivation is $\mathrm{HO}\pi_{\mathrm{ref}}$; the subset without passivation and refusal is $\mathrm{HO}\pi$. These are the *higher-order* concurrent languages. In a *first-order* concurrent language, in contrast, all channels and prefixes are first-order (i.e., unit is the only communicable value) and the passivation prefix is disallowed. The resulting language is

$\mathrm{CCS}^-_{\mathrm{ref}}$; when also refusal is disallowed, the language is CCS^-. (The '-' sign emphasizes the lack of the choice operator; see however Section 7.3.)

As usual, for any language, say $\mathtt{AL}$, we write $\mathtt{AL}(\mathcal{L})$ for the extension of $\mathtt{AL}$ with the first-order processes from the LTS or RPLTS $\mathcal{L}$, i.e., with the additional grammar production

$$M := \ldots \;\Big|\; P$$

where P is an $\mathcal{L}$ process. In the languages without passivation, the presence of kells is irrelevant; e.g., a process $N \mid [\![M]\!]_l$ behaves like $N \mid M$.

To avoid run-time errors in interactions, we assume a basic type system, that distinguishes three types of values: the unit value $\star$, the set $Pr_{\mathcal{L}}$ of tested $\mathcal{L}$ processes, and arbitrary processes Pr_{all}, with the subtyping $Pr_{\mathcal{L}} \leq Pr_{\mathrm{all}}$. As a consequence, there are three types of names and variables. We distinguish the tested $\mathcal{L}$ processes from arbitrary processes because we allow refusal to act only on the former processes (this simplifies the operational rules, though it is not essential). For lack of space, we omit the (expected) typing rules.

4.1 Fully Nondeterministic Processes

The operational rules for the full language $\mathrm{HO}\pi_{\mathtt{pass},\mathrm{ref}}(\mathcal{L})$, when $\mathcal{L}$ is an LTS, are presented in Figure 4. The grammar for action labels is:

$$\mu := \tau \;\Big|\; \overline{a}\langle M\rangle \;\Big|\; a\langle M\rangle \;\Big|\; \omega \;\Big|\; \overline{\mathtt{pass}}_l M \;\Big|\; \mathtt{pass}_l N \;\Big|\; \overline{\widetilde{r}}_l \;\Big|\; \widetilde{r}_l$$

where $\overline{a}\langle M\rangle$, $\overline{\mathtt{pass}}_l M$, and $\overline{\widetilde{r}}_l$ are the dual of, and synchronize with, $a\langle M\rangle$, $\mathtt{pass}_l N$, and $\widetilde{r}_l$. The dual of μ is $\overline{\mu}$. In the languages without refusal ($\mathrm{HO}\pi_{\mathtt{pass}}(\mathcal{L})$, $\mathrm{HO}\pi(\mathcal{L})$, $\mathrm{CCS}^-(\mathcal{L})$), rules RefLoc and RefPre are missing; in the languages without passivation ($\mathrm{HO}\pi_{\mathrm{ref}}(\mathcal{L})$, $\mathrm{HO}\pi(\mathcal{L})$, $\mathrm{CCS}^-(\mathcal{L})$, $\mathrm{CCS}^-_{\mathrm{ref}}(\mathcal{L})$), rules PassLoc and PassPre are missing. Further, in the CCS languages the only value exchanged is $\star$.

4.2 Reactive Probabilistic Processes

If $\mathcal{L}$ is an RPLTS, the rules for parallel composition (Figure 5) propagate the probability distributions reached from the processes in $\mathcal{L}$. For any Δ_1, Δ_2, we define the distribution:

$$\Delta_1 \mid \Delta_2(M) \stackrel{\text{def}}{=} \begin{cases} \Delta_1(M_1) \cdot \Delta_2(M_2) & \text{if } M = M_1 \mid M_2 \\ 0 & \text{otherwise} \end{cases}$$

Differently from the contexts of λ-calculi, now the contexts are nondeterministic. Therefore, in general $C[P]$ is a nondeterministic *and* probabilistic term (i.e., an NPLTS).

5. CCS Languages: Separation Results

In the results on the concurrent languages, we include, in parenthesis, reference to the results for the λ-calculi to ease the comparison.

5.1 Fully Nondeterministic Processes

When $\mathcal{L}$ is an LTS, $\simeq^{\mathcal{L}}_{\mathrm{CCS}^-}$ coincides with ordinary may-testing equivalence [7] and hence with $\sim_{\mathrm{Tr}}$, because the canonical tests of [7] can be encoded without resorting to the choice operator. For a similar reason, $\simeq^{\mathcal{L}}_{\mathrm{CCS}^-_{\mathrm{ref}}}$ coincides with the refusal testing equivalence of [23] and hence with $\sim_{\mathrm{FTr}}$.

5.2 Reactive Probabilistic Processes

When $\mathcal{L}$ is an RPLTS, the contextual equivalences induced by CCS^- and $\mathrm{CCS}^-_{\mathrm{ref}}$ are comprised between $\sim_{\mathrm{PB}}$ and $\sim_{\mathrm{PFTr}}$.

THEOREM 5.1. If $\mathcal{L}$ is an RPLTS, then:

$$\sim_{\mathrm{PB}} \subsetneq \simeq^{\mathcal{L}}_{\mathrm{CCS}^-_{\mathrm{ref}}} \subsetneq \simeq^{\mathcal{L}}_{\mathrm{CCS}^-} \subsetneq \sim_{\mathrm{PFTr}} \;\; (= \simeq^{\mathcal{L}}_{\mathrm{K}\Lambda_{\mathrm{N}}})$$

Proof [Sketch] To prove the first inclusion, we exploit the congruence of $\sim_{\mathrm{PB}}$, and we observe that a probabilistic bisimulation relating $C[P_1]$ and $C[P_2]$ induces projections that are probabilistic bisimulations over pairs of maximal resolutions. Corresponding maximal resolutions turn out to have the same success probability, from which the result follows. The second inclusion is immediate. To prove the third inclusion, given an arbitrary failure trace ϕ of length n, we consider the failure trace ϕ^ω obtained from ϕ by replacing every pair (a_i, F_i), $2 \leq i \leq n$, with (a_i^ω, F_i). Moreover, starting from the processes P_1 and P_2 under consideration, we relabel with a_i^ω each transition executable after i steps, $2 \leq i \leq n$, which departs from a process that does not enable any action in F_{i-1}. The result stems from the fact that the relabeling proceeds in the same way for P_1 and P_2 because, should this not be the case at step i, then the test with a single computation labeled with the first $i-1$ coactions and terminating with the parallel composition of the various coactions in F_{i-1} followed by ω would separate the two processes.

The inclusion of $\simeq^{\mathcal{L}}_{\mathrm{CCS}^-}$, and hence of $\simeq^{\mathcal{L}}_{\mathrm{CCS}^-_{\mathrm{ref}}}$, in $\sim_{\mathrm{PFTr}}$ is strict: the $\sim_{\mathrm{PFTr}}$-equivalent processes

$$P \stackrel{\text{def}}{=} a.((b.d + c.e) +_{0.5} (b.f + c.g))$$
$$P' \stackrel{\text{def}}{=} a.(b.(d +_{0.5} f) + c.(e +_{0.5} g))$$

are distinguished by the CCS^- context $[\cdot] \mid \overline{a}.(\overline{b}.\overline{d}.\omega \mid \overline{c}.\overline{g}.\omega)$ (calling C this context, the maximum probability of succeeding for $C[P]$ is 1, whereas that of $C[P']$ is 0.5). The inclusion of $\simeq^{\mathcal{L}}_{\mathrm{CCS}^-_{\mathrm{ref}}}$ in $\simeq^{\mathcal{L}}_{\mathrm{CCS}^-}$ is strict as well: the processes Q, Q' in Example 3.4 are not distinguished in CCS^-, but they are distinguished by the $\mathrm{CCS}^-_{\mathrm{ref}}$ context $[\![[\cdot]]\!]_l \mid \overline{a}.(\overline{b}.\overline{c}.\omega \mid \overline{b}.\widetilde{c}_l.\omega)$ (the maximum probabilities of success are respectively 1 and 0.5).

Finally, probabilistic bisimilarity is strictly included in $\simeq^{\mathcal{L}}_{\mathrm{CCS}^-_{\mathrm{ref}}}$, since the non probabilistic bisimilar processes

$$R \stackrel{\text{def}}{=} d.(e.Q +_{0.5} e.Q')$$
$$R' \stackrel{\text{def}}{=} d.e.(Q +_{0.5} Q')$$

cannot be distinguished by any $\mathrm{CCS}^-_{\mathrm{ref}}$-context. The different timing of the initial choices in R and R' is visible under bisimilarity but is not under the may semantics. (Intuitively, in the may semantics Q and Q' can only be distinguished by tests that exhibit success probabilities 1 and 0.5, respectively; we would need however a richer range of success probabilities to be able to separate R and R'.) □

6. HOπ Languages: Separation Results

6.1 Fully Nondeterministic Processes

The proof schemata for Theorem 6.1 are as those for the analogous results in λ-calculi; in one direction we essentially encode the (higher-order) separating tests of the λ-calculi into the Higher-Order π-calculi.

THEOREM 6.1. If $\mathcal{L}$ is an image-finite LTS, then:

1. $\simeq^{\mathcal{L}}_{\mathrm{HO}\pi_{\mathtt{pass},\mathrm{ref}}} = \sim_{\mathrm{RS}} \;\; (= \simeq^{\mathcal{L}}_{\mathrm{K}\Lambda_{\mathrm{V}}})$;
2. $\simeq^{\mathcal{L}}_{\mathrm{HO}\pi_{\mathrm{ref}}} = \sim_{\mathrm{FTr}} \;\; (= \simeq^{\mathcal{L}}_{\mathrm{K}\Lambda_{\mathrm{N}}})$;
3. $\simeq^{\mathcal{L}}_{\mathrm{HO}\pi_{\mathtt{pass}}} = \sim_{\mathrm{S}} \;\; (= \simeq^{\mathcal{L}}_{\mathrm{K}\Lambda_{\mathrm{V}-\mathrm{ref}}})$;
4. $\simeq^{\mathcal{L}}_{\mathrm{HO}\pi} = \sim_{\mathrm{Tr}} \;\; (= \simeq^{\mathcal{L}}_{\mathrm{K}\Lambda_{\mathrm{N}-\mathrm{ref}}})$. ■

EXAMPLE 6.2. In this example, we test a process P by placing it in a kell and then running a test M in parallel, as in $[\![P]\!]_l \mid M$. The tests $\overline{a}.\omega$ and $\widetilde{a}_l.\omega$ check whether $P \xrightarrow{a}$ and $P \not\xrightarrow{a}$, respectively. The test M'_1, below, in $\mathrm{HO}\pi_{\mathrm{ref}}$ performs the same test as the term M_1 in $\mathrm{K}\Lambda_{\mathrm{N}}$ discussed in Example 3.1, while M'_2 in $\mathrm{HO}\pi_{\mathtt{pass}}$

$$
\begin{array}{rcll}
u & ::= & a \mid l & \text{(names)} \\
r & ::= & a \mid \overline{a} & \text{(input/output channels)} \\
\alpha & ::= & a(x) \mid \overline{a}\langle M\rangle \mid \omega \mid \mathtt{pass}_l(x) \mid \widetilde{r}_l & \text{(prefixes)} \\
M & ::= & M \mid M \mid \alpha.M \mid x \mid \mathbf{0} \mid [\![M]\!]_l \mid \star & \text{(processes and values)}
\end{array}
$$

Figure 3. The syntax for HO$\pi_{\mathtt{pass},\mathrm{ref}}$

$$
\textsc{ParL}\ \frac{M_1 \xrightarrow{\mu} M_1'}{M_1 \mid M_2 \xrightarrow{\mu} M_1' \mid M_2} \qquad
\textsc{ParR}\ \frac{M_2 \xrightarrow{\mu} M_2'}{M_1 \mid M_2 \xrightarrow{\mu} M_1 \mid M_2'} \qquad
\textsc{Com}\ \frac{M_1 \xrightarrow{\mu} M_1 \quad M_2 \xrightarrow{\overline{\mu}} M_2'}{M_1 \mid M_2 \xrightarrow{\tau} M_1' \mid M_2'} \qquad
\textsc{foPr}\ \frac{P \xrightarrow{r} P' \text{ in } \mathcal{L}}{P \xrightarrow{r\langle\star\rangle} P'}
$$

$$
\textsc{Inp}\ \frac{}{a(x).M \xrightarrow{a\langle N\rangle} M\{^N\!/_x\}} \qquad
\textsc{Out}\ \frac{}{\overline{a}\langle N\rangle.M \xrightarrow{\overline{a}\langle N\rangle} M} \qquad
\textsc{RefLoc}\ \frac{P \not\xrightarrow{r} \text{ in } \mathcal{L}}{[\![P]\!]_l \xrightarrow{\widetilde{r}_l} [\![P]\!]_l} \qquad
\textsc{RefPre}\ \frac{}{\widetilde{r}_l.N \xrightarrow{\widetilde{r}_l} N}
$$

$$
\textsc{Succ}\ \frac{}{\omega.M \xrightarrow{\omega} M} \qquad
\textsc{Kell}\ \frac{M \xrightarrow{\mu} M'}{[\![M]\!]_l \xrightarrow{\mu} [\![M']\!]_l} \qquad
\textsc{PassLoc}\ \frac{}{[\![N]\!]_l \xrightarrow{\overline{\mathtt{pass}_l}\,N} \mathbf{0}} \qquad
\textsc{PassPre}\ \frac{}{\mathtt{pass}_l(x).M \xrightarrow{\mathtt{pass}_l\,N} M\{^N\!/_x\}}
$$

Figure 4. The operational semantics for HO$\pi_{\mathtt{pass},\mathrm{ref}}(\mathcal{L})$

$$
\textsc{ParL}\ \frac{M_1 \xrightarrow{\mu} \Delta_1}{M_1 \mid M_2 \xrightarrow{\mu} \Delta_1 \mid \overline{M_2}} \qquad
\textsc{ParR}\ \frac{M_2 \xrightarrow{\mu} \Delta_2}{M_1 \mid M_2 \xrightarrow{\mu} \overline{M_1} \mid \Delta_2} \qquad
\textsc{Com}\ \frac{M_1 \xrightarrow{\mu} \Delta_1 \quad M_2 \xrightarrow{\overline{\mu}} \Delta_2}{M_1 \mid M_2 \xrightarrow{\tau} \Delta_1 \mid \Delta_2}
$$

Figure 5. The rules for parallel composition in the probabilistic setting

corresponds to M_2 in $\mathtt{K}\Lambda_{\mathrm{V}-\mathrm{ref}}$ (the second occurrence $\mathtt{pass}_l(y)$ of the passivation operator destroys the first copy of the tested process, so as to ensure that the test $\overline{c}.\omega$ is executed on the second copy).

$$
\begin{array}{rcl}
M_1' & \stackrel{\text{def}}{=} & \overline{a}.\widetilde{c}_l.\overline{b}.\omega \\
M_2' & \stackrel{\text{def}}{=} & \overline{a}.\mathtt{pass}_l(x).([\![x]\!]_l \mid \overline{b}.\mathtt{pass}_l(y).([\![x]\!]_l \mid \overline{c}.\omega))
\end{array}
$$

■

6.2 Reactive Probabilistic Processes

The proof of Theorem 6.3(1) is analogous to the proofs for the λ-calculi $\mathtt{K}\Lambda_{\mathrm{V}}$ and $\mathtt{K}\Lambda_{\mathrm{V}-\mathrm{ref}}$. Again, in one direction we essentially encode the separating tests of the λ-calculi. For the opposite direction, we use a fixed point-point approach, but unlike in λ-calculi, now the definition of may-equivalence through suprema (Definition 2.10, following [13, 33], as opposed to the definitions in [8, 28]) is important, because of a potentially infinite number of resolutions.

THEOREM 6.3. If $\mathcal{L}$ is an RPLTS, then:

1. $\simeq^{\mathcal{L}}_{\mathrm{HO}\pi_{\mathtt{pass},\mathrm{ref}}} = \simeq^{\mathcal{L}}_{\mathrm{HO}\pi_{\mathtt{pass}}} = \sim_{\mathrm{PB}} (= \simeq^{\mathcal{L}}_{\mathtt{K}\Lambda_{\mathrm{V}}} = \simeq^{\mathcal{L}}_{\mathtt{K}\Lambda_{\mathrm{V}-\mathrm{ref}}})$;
2. $\simeq^{\mathcal{L}}_{\mathrm{HO}\pi_{\mathrm{ref}}} \subseteq \simeq^{\mathcal{L}}_{\mathrm{HO}\pi} \subsetneq \simeq^{\mathcal{L}}_{\mathrm{CCS}^-}$;
3. $\simeq^{\mathcal{L}}_{\mathrm{HO}\pi_{\mathrm{ref}}} \subsetneq \simeq^{\mathcal{L}}_{\mathrm{CCS}^-_{\mathrm{ref}}}$.

■

The following processes witness the strictness of the inclusion of $\simeq^{\mathcal{L}}_{\mathrm{HO}\pi}$ in $\simeq^{\mathcal{L}}_{\mathrm{CCS}^-}$:

$$
\begin{array}{rcl}
P & \stackrel{\text{def}}{=} & d.Q + e.(f +_{0.5} \mathbf{0}) \\
P' & \stackrel{\text{def}}{=} & d.Q' + e.(f +_{0.5} \mathbf{0})
\end{array}
$$

for Q, Q' as in Example 3.4. Processes P, P', different under $\sim_{\mathrm{PB}}$, are identified by $\simeq^{\mathcal{L}}_{\mathrm{CCS}^-}$. They are also separated in HOπ, via the context

$$
C \stackrel{\text{def}}{=} \overline{h}\langle[\cdot]\rangle \mid h(x).(x \mid \overline{d}.\overline{a}.(\overline{b}.\overline{c}.\omega \mid \overline{b}.(x \mid \overline{e}) \mid \overline{f}.\omega)).
$$

Intuitively, this context uses higher-order communication to make copies of the tested process at the beginning, and then exploits the right-hand branch $e.(f +_{0.5} \mathbf{0})$ of the process itself in order to test the left-hand branch.

Analogously, let R, R' be as in Section 5.2 and define the processes

$$
\begin{array}{rcl}
S & \stackrel{\text{def}}{=} & R + f.(g +_{0.5} \mathbf{0}) + h.(i +_{0.6} \mathbf{0}) \\
S' & \stackrel{\text{def}}{=} & R' + f.(g +_{0.5} \mathbf{0}) + h.(i +_{0.6} \mathbf{0}).
\end{array}
$$

It follows from $R \simeq^{\mathcal{L}}_{\mathrm{CCS}^-_{\mathrm{ref}}} R'$ that $S \simeq^{\mathcal{L}}_{\mathrm{CCS}^-_{\mathrm{ref}}} S'$, while the HO$\pi$-context C' distinguishes S and S'

$$
\begin{array}{rcl}
C' & \stackrel{\text{def}}{=} & \overline{j}\langle[\cdot]\rangle \mid j(x).(x \mid \overline{d}.(\overline{e}.\overline{a}.T \mid \overline{e}.(x \mid \overline{h}.\overline{i}.\omega))) \\
T & \stackrel{\text{def}}{=} & \overline{b}.\overline{c}.\omega \mid \overline{b}.(x \mid \overline{f}.\overline{g}.\omega).
\end{array}
$$

Since HO$\pi \subseteq$ HOπ_{ref}, the example shows the strictness of the inclusion of $\simeq^{\mathcal{L}}_{\mathrm{HO}\pi_{\mathrm{ref}}}$ in $\simeq^{\mathcal{L}}_{\mathrm{CCS}^-_{\mathrm{ref}}}$.

7. Extensions and Variations

7.1 Global vs. Local Communications

In our concurrent languages, communications are *network transparent*, in the sense that they take place irrespective of the locations in which the interacting processes are placed. In the literature, this approach to communication is sometimes called *global*, as opposed to the *local* approach, where communication is subject to physical proximity. Under local communications, locations form communication barriers, because they confine where interactions can occur, with a finer control over communication interferences. We have checked that this extra precision in communications does not affect the discriminating power of the languages, when formalizing local communications as in the Kell calculus [27].

7.2 May vs. Must Equivalences

In the contextual equivalences we have used, success is in the 'may' style. The 'must' variants focus on the success of maximal τ-computations (i.e., whose steps are all labeled with τ). In the LTS case, with respect to Definition 2.8 the preorder $\leq^{\mathcal{L}}_{\mathtt{AL},\mathrm{must}}$ is introduced by requiring that, if all maximal τ-computations from $C[P_1]$ are successful, then so are those from $C[P_2]$. In the RPLTS case, the definition of $\leq^{\mathcal{L}}_{\mathtt{AL},\mathrm{must}}$ is obtained from Definition 2.10 by simply

using $\sqcap$ in place of $\sqcup$, i.e., by considering the minimum probability of reaching success in the various maximal τ-resolutions.

The must-equivalences coincide with the may-equivalences when the tested processes are RPLTSs and the testing language is a λ-calculus, because internal nondeterminism does not occur. In contrast, nondeterminism may spring up when the testing language is concurrent, or when the tested processes are LTSs rather than RPLTSs. We discuss below a few scenarios in which must- and may-equivalences coincide despite the presence of nondeterminism.

Let us start with LTSs. Due to the absence of divergence, in $\mathrm{CCS}^{-}_{\mathrm{ref}}$ the must-equivalence coincides with the refusal testing equivalence of [23] and hence with $\sim_{\mathrm{FTr}}$; thus, it coincides with the may-equivalence. In contrast, it follows from [6] that in CCS^{-} the must-equivalence coincides with $\sim_{\mathrm{F}}$ and hence is strictly finer than the may-equivalence, which is $\sim_{\mathrm{Tr}}$. However, we can show that, if the testing language is $\mathtt{K}\Lambda_{\mathrm{V}}$ or $\mathrm{HO}\pi_{\mathtt{pass},\mathrm{ref}}$, then also the reverse inclusion holds.

THEOREM 7.1. If $\mathcal{L}$ is an LTS, then $\leq^{\mathcal{L}}_{\mathtt{AL}} \subseteq \geq^{\mathcal{L}}_{\mathtt{AL},\mathrm{must}}$ for $\mathtt{AL} \in \{\mathtt{K}\Lambda_{\mathrm{V}}, \mathrm{HO}\pi_{\mathtt{pass},\mathrm{ref}}\}$. ■

COROLLARY 7.2. If $\mathcal{L}$ is an LTS, then $\simeq^{\mathcal{L}}_{\mathtt{AL},\mathrm{must}} = \simeq^{\mathcal{L}}_{\mathtt{AL}} = \sim_{\mathrm{RS}}$ for $\mathtt{AL} \in \{\mathtt{K}\Lambda_{\mathrm{V}}, \mathrm{HO}\pi_{\mathtt{pass},\mathrm{ref}}\}$. ■

The refusal operator (respectively, the REFACT rule in λ-calculus) is essential for the reverse inclusion to hold: the processes $P \stackrel{\mathrm{def}}{=} a.b + a$ and $P' \stackrel{\mathrm{def}}{=} a.b$ are $\sim_{\mathrm{S}}$-equivalent and hence may-equivalent both in $\mathtt{K}\Lambda_{\mathrm{V-ref}}$ and in $\mathrm{HO}\pi_{\mathtt{pass}}$, but only P' always succeeds when the trace $a\,b$ is tested.

We now move to RPLTSs. By Theorem 3.3, the tests needed in order to distinguish $\sim_{\mathrm{PB}}$-inequivalent RPLTSs are encodable in $\mathtt{K}\Lambda_{\mathrm{V-ref}}$. The passivation operator allows us to encode these tests in $\mathrm{HO}\pi_{\mathtt{pass}}$ without losing the sequentiality of the tests. Since RPLTSs do not have internal nondeterminism and the tests are sequential, the resulting NPLTS has a unique maximal resolution. Hence, $\sim_{\mathrm{PB}}$-inequivalent RPLTSs are neither may-equivalent nor must-equivalent.

THEOREM 7.3. If $\mathcal{L}$ is an RPLTS, then $\simeq^{\mathcal{L}}_{\mathtt{AL},\mathrm{must}} = \simeq^{\mathcal{L}}_{\mathtt{AL}} = \sim_{\mathrm{PB}}$ for $\mathtt{AL} \in \{\mathrm{HO}\pi_{\mathtt{pass}}, \mathrm{HO}\pi_{\mathtt{pass},\mathrm{ref}}\}$. ■

7.3 Other Operators

The concurrent calculi we have considered do not include certain common operators, such as restriction, recursion, relabeling, and choice, so to make the operational rules simpler or because often omitted in higher-order languages. The addition of these operators would not change the results presented (we assume that the hole of a context is not allowed to occur in recursive definitions). Some care is necessary with restriction in the presence of passivation, along the lines of [15, 24], because the lazy scope extrusion on restriction could allow contextual equivalence to make distinctions on processes solely on the basis of their free names [21].
In HOπ we only allow communication of processes; the addition of process *abstractions*, or the name-passing communications of the π-calculus, as in the full HOπ, would not affect the results.

8. Conclusions and Future Work

We have studied the discriminating power offered by *higher-order concurrent* languages such as HOπ, without and with passivation, and contrasted it with those offered by *higher-order sequential* languages à la λ-calculus and by *first-order concurrent* languages à la CCS. We have measured this discriminating power on the basis of the distinctions that the languages, possibly extended with refusal, allow us to make on first-order processes that are either fully nondeterministic (LTSs) or reactive probabilistic (RPLTSs).

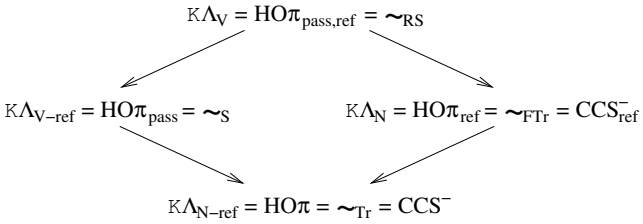

Figure 6. The spectrum of equivalences for fully nondeterministic processes

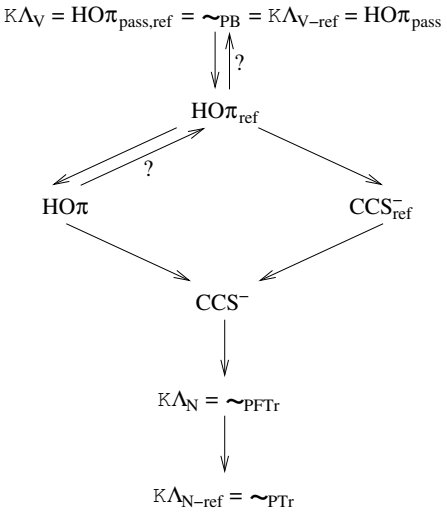

Figure 7. The spectrum of equivalences for reactive probabilistic processes

The discriminating power of HOπ with passivation coincides with that of the call-by-value λ-calculus (with or without passivation), both on LTSs and on RPLTSs. Intuitively, HOπ with passivation and the call-by-value λ-calculus are both capable of implementing the 'and' of two tests. That is, the equivalence induced by these languages are characterized by modal logics that include the 'and' connective between formulas and are therefore 'branching-sensitive'.

The addition of refusal increases the discriminating power of all the considered languages when testing LTSs. This is not always the case on RPLTSs. One reason is that, similarly to fully probabilistic processes [12, 14], the spectrum of equivalences for RPLTSs is narrower than for LTSs.

On LTSs, the extra discriminating power offered in concurrency by passivation over higher-order communication corresponds, in λ-calculi, to the call-by-value possibility of reducing the argument of a function and then capturing the result. The addition to the λ-calculus of a pure sequencing construct $M_1; M_2$ akin to command sequentialization of imperative languages would not affect the results; in contrast, parametrized sequencing, e.g., $\mathtt{pass}(x)\ \mathtt{in}\ M$, would bridge the gap between call-by-name and call-by-value.

On RPLTSs, we do not know exactly what are the equivalences induced by CCS^{-} and $\mathrm{CCS}^{-}_{\mathrm{ref}}$, though we know they are strictly in between probabilistic failure-trace equivalence and probabilistic bisimilarity. We are not aware of RPLTS equivalences in the literature with the same property. The lack of any copying facility makes the CCS^{-} equivalence also strictly coarser than those of HOπ and

of all other concurrent languages considered. Another question that remains is whether the equivalences induced by HOπ (with or without refusal) coincide, and whether they are strictly coarser than probabilistic bisimilarity.

Figures 6 and 7 summarize the relationship among the various equivalences on a first-order LTS or RPLTS, respectively, that have been considered in the paper. In the figures, the name of a language, say AL, stands for the contextual equivalence $\simeq^{\mathcal{L}}_{\mathtt{AL}}$. A single arrow denotes a strict inclusion, unless the arrow is coupled with the reverse arrow and the question mark, in which case we do not know whether the inclusion is strict or not.

In the paper we have used LTSs/RPLTSs without internal moves, which means that the induced equivalences are 'strong', and we have looked at 'may' forms of contextual equivalence, discussing only a few instances of 'must' equivalence. Two natural developments of this work are thus (i) to systematically address the must-equivalences and (ii) to admit internal actions in the tested processes and therefore move to 'weak' behavioural relations.

We have admitted probabilities in the tested first-order processes, but not in the testing languages. It would be interesting to see if and how the addition of probabilities to the testing languages affects the results. Likewise, we have examined the equivalences induced on purely nondeterministic processes (LTSs), and on reactive probabilistic processes (RPLTSs), but we have not considered combinations of them; this would amount to studying whether the contextual equivalences induced on NPLTSs coincide with known probabilistic testing equivalences, e.g., [8, 9, 28] (characterized also as variants of simulation), or other behavioural relations are needed.

Acknowledgments

The authors acknowledge support from the MIUR-PRIN project 'CINA', and the ANR project 12IS02001 'PACE'.

References

[1] F. Bartels. GSOS for probabilistic transition systems. In *Proc. CMCS'02*, ENTCS 65(1):29–53. Elsevier, 2002

[2] B. Bloom, S. Istrail, and A.R. Meyer. Bisimulation can't be traced. *Journal of the ACM*, 42:232–268, 1995.

[3] S.D. Brookes, C.A.R. Hoare, and A.W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31:560–599, 1984.

[4] G. Castagna, J. Vitek, and F. Zappa Nardelli. The Seal calculus. *Inf. Comput.*, 201(1):1–54, 2005.

[5] P.R. D'Argenio and M.D. Lee. Probabilistic transition system specification: Congruence and full abstraction of bisimulation. In *Proc. FoSSaCS'12*, LNCS 7213, pages 452–466. Springer, 2012.

[6] R. De Nicola. Extensional equivalences for transition systems. *Acta Informatica*, 24:211–237, 1987.

[7] R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Comp. Sci.*, 34:83–133, 1984.

[8] Y. Deng, R.J. van Glabbeek, M. Hennessy, and C. Morgan. Characterising testing preorders for finite probabilistic processes. *Logical Methods in Computer Science*, 4(4:4):1–33, 2008.

[9] Y. Deng, R.J. van Glabbeek, M. Hennessy, and C. Morgan. Testing finitary probabilistic processes. In *Proc. CONCUR'09*, LNCS 5710, pages 274–288. Springer, 2009.

[10] J.C. Godskesen and T.T. Hildebrandt. Extending Howe's method to early bisimulations for typed mobile embedded resources with local names. In *Proc. FSTTCS'05*, LNCS 3821, pages 140–151. Springer, 2005.

[11] J.F. Groote and F.W. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Inf. Comp.*, 100:202–260, 1992.

[12] B. Jonsson and K.G. Larsen. Specification and refinement of probabilistic processes. In *Proc. LICS'91*, pages 266–277. IEEE, 1991.

[13] B. Jonsson and W. Yi. Compositional testing preorders for probabilistic processes. In *Proc. LICS'95*, pages 431–441. IEEE, 1995.

[14] C.-C. Jou and S.A. Smolka. Equivalences, congruences, and complete axiomatizations for probabilistic processes. In *Proc. CONCUR'90*, LNCS 458, pages 367–383. Springer, 1990.

[15] V. Koutavas and M. Hennessy. Symbolic bisimulation for a higher-order distributed language with passivation. In *Proc. CONCUR'13*, LNCS 8052, pages 167–181. Springer, 2013.

[16] M. Kwiatkowska and G. Norman. A testing equivalence for reactive probabilistic processes. In *Proc. EXPRESS'98*, ENTCS 16(2), pages 114–132. Elsevier, 1998.

[17] R. Lanotte and S. Tini. Probabilistic bisimulation as a congruence. *ACM Trans. Comput. Log.*, 10(2), 2009.

[18] K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Inf. Comput.*, 94:1–28, 1991.

[19] S. Lenglet, A. Schmitt, and J.-B. Stefani. Howe's method for calculi with passivation. In *Proc. CONCUR'09*, LNCS 5710, pages 448–462. Springer, 2009.

[20] S. Lenglet, A. Schmitt, and J.-B. Stefani. Normal bisimulations in calculi with passivation. In *Proc. FoSSaCS'09*, LNCS 5504, pages 257–271. Springer, 2009.

[21] S. Lenglet, A. Schmitt, and J.-B. Stefani. Characterizing contextual equivalence in calculi with passivation. *Inf. Comput.*, 209(11), 2011.

[22] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[23] I. Phillips. Refusal testing. *Theoretical Comp. Sci.*, 50:241–284, 1987.

[24] A. Piérard and E. Sumii. A higher-order distributed calculus with name creation. In *Proc. LICS'12*, pages 531–540. IEEE, 2012.

[25] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis CST–99–93, Department of Computer Science, University of Edinburgh, 1992.

[26] A. Schmitt and J.-B. Stefani. The M-calculus: a higher-order distributed process calculus. In *Proc. POPL'03*, pages 50–61. ACM, 2003.

[27] A. Schmitt and J.-B. Stefani. The kell calculus: A family of higher-order distributed process calculi. In *Proc. GC'04*, LNCS 3267, pages 146–178. Springer, 2005.

[28] R. Segala. Testing probabilistic automata. In *Proc. CONCUR'96*, LNCS 1119, pages 299–314. Springer, 1996.

[29] P. Sewell, J.J. Leifer, K. Wansbrough, F. Zappa Nardelli, M. Allen-Williams, P. Habouzit, and V. Vafeiadis. Acute: High-level programming language design for distributed computation. *J. Funct. Program.*, 17(4-5):547–612, 2007.

[30] B. Thomsen. Plain CHOCS, a second generation calculus for higher-order processes. *Acta Informatica*, 30:1–59, 1993.

[31] F. van Breugel, M. Mislove, J. Ouaknine, and J. Worrell. Domain theory, testing and simulation for labelled Markov processes. *Theoretical Comp. Sci.*, 333:171–197, 2005.

[32] R.J. van Glabbeek. The linear time – branching time spectrum I. In *Handbook of Process Algebra*, pages 3–99. Elsevier, 2001.

[33] W. Yi and K.G. Larsen. Testing probabilistic and nondeterministic processes. In *Proc. PSTV'92*, pages 47–61. North-Holland, 1992.